**Kubernetes and Jenkins Deployment Project Documentation**

**Project 1: Kubernetes Mini Demo (Without Jenkins)**

**Overview**

This project focused on **deploying a simple Flask application** inside a **Kubernetes cluster running on Minikube**. The objective was to **containerize a Python Flask app** and deploy it in Kubernetes while learning about **Deployments, Services, ConfigMaps, and Namespaces**.

This project **does not use Jenkins**—instead, the deployment process is **manual**, where we build Docker images, apply Kubernetes manifests, and troubleshoot issues manually.

**Project Structure**

mini-k8s-demo/

│── app/

│ ├── app.py # Flask Application

│ ├── requirements.txt # Dependencies

│ ├── Dockerfile # Docker Configuration

│── k8s/

│ ├── namespace.yaml # Kubernetes Namespace

│ ├── configmap.yaml # Kubernetes ConfigMap

│ ├── deployment.yaml # Kubernetes Deployment

│ ├── service.yaml # Kubernetes Service

│── deploy.sh # Deployment Script

│── troubleshoot.sh # Troubleshooting Script

│── k8s-explained.md # Kubernetes Concepts Explained

**Key Learnings and Implementation Steps**

**1️.Dockerizing the Flask Application**

* We created a **simple Flask app (app.py)** that exposes three API endpoints:
  + / → Shows pod and service information.
  + /api/info → Returns application metadata.
  + /api/health → Health check endpoint for Kubernetes liveness/readiness probes.
* **Created a requirements.txt** to define the necessary dependencies (Flask).
* **Wrote a Dockerfile** to containerize the app:
* FROM python:3.9-slim
* WORKDIR /app
* COPY requirements.txt .
* RUN pip install --no-cache-dir -r requirements.txt
* COPY app.py .
* EXPOSE 5000
* CMD ["python", "app.py"]

**Key Takeaways from the Dockerfile:**

* + Used python:3.9-slim to keep the image size small.
  + Installed dependencies using pip install -r requirements.txt.
  + Set the default command to run Flask when the container starts.
  + Exposed **port 5000**, so Kubernetes can access the service.

**2️.Writing Kubernetes YAML Configurations**

Each Kubernetes resource was defined in a YAML file.

**Namespace (namespace.yaml)**

Namespaces help isolate applications within Kubernetes.

apiVersion: v1

kind: Namespace

metadata:

name: mini-demo

**ConfigMap (configmap.yaml)**

ConfigMaps store environment variables, allowing us to separate configuration from code.

apiVersion: v1

kind: ConfigMap

metadata:

name: app-config

namespace: mini-demo

data:

APP\_NAME: "Kubernetes Mini Demo"

APP\_VERSION: "1.0.0"

**Deployment (deployment.yaml)**

A **Deployment** ensures the application runs in a **replicated and self-healing** manner.

apiVersion: apps/v1

kind: Deployment

metadata:

name: flask-app

namespace: mini-demo

spec:

replicas: 2

selector:

matchLabels:

app: flask-app

template:

metadata:

labels:

app: flask-app

spec:

containers:

- name: flask-app

image: mini-k8s-demo:latest

imagePullPolicy: Never

ports:

- containerPort: 5000

envFrom:

- configMapRef:

name: app-config

* **Replicas = 2** ensures **high availability**.
* imagePullPolicy: Never ensures Kubernetes uses the **locally built Docker image**.
* **ConfigMap values** are injected as environment variables.
* **Health checks** ensure Kubernetes knows when a pod is ready or needs a restart.

**Service (service.yaml)**

A **Kubernetes Service** exposes our Flask app **internally and externally**.

apiVersion: v1

kind: Service

metadata:

name: flask-service

namespace: mini-demo

spec:

type: NodePort

selector:

app: flask-app

ports:

- port: 80

targetPort: 5000

nodePort: 30080

* **NodePort service** makes the application accessible externally.
* **30080** → The external port that users can access the application.

**3️.Automating Deployment with deploy.sh**

We created a **Bash script** to automate the Kubernetes deployment process.

#!/bin/bash

set -e

echo "🚀 Starting Minikube..."

minikube start --driver=docker

echo "🐳 Setting up Docker..."

eval $(minikube docker-env)

echo "🛠️ Building Docker Image..."

docker build -t mini-k8s-demo:latest app/

echo "📌 Deploying to Kubernetes..."

kubectl apply -f k8s/namespace.yaml

kubectl apply -f k8s/configmap.yaml

kubectl apply -f k8s/deployment.yaml

kubectl apply -f k8s/service.yaml

echo "✅ Deployment complete! Access the app using:"

minikube service flask-service -n mini-demo --url

**Steps Automated:**

1. **Checks if Minikube is running**.
2. **Sets up Docker inside Minikube**.
3. **Builds the Docker image**.
4. **Deploys Kubernetes manifests**.
5. **Fetches the service URL**.

**4️.Troubleshooting Common Issues**

To debug Kubernetes and Minikube, we created a **troubleshoot.sh** script.

#!/bin/bash

echo "🔍 Checking Minikube Status..."

minikube status

echo "🛠️ Checking Kubernetes Resources..."

kubectl get pods -n mini-demo

kubectl get services -n mini-demo

kubectl logs -l app=flask-app -n mini-demo

If the pod is not running:

kubectl describe pod <pod-name> -n mini-demo

kubectl logs <pod-name> -n mini-demo

If Minikube is down:

minikube stop && minikube delete

minikube start --driver=docker

**Final Outcome**

By running:

chmod +x deploy.sh

./deploy.sh

We successfully deployed a **Flask application running inside a Kubernetes cluster** using **Minikube**.

**Project 2: Kubernetes with Jenkins (CI/CD Pipeline)**

**Overview**

This project builds upon the first project by introducing **Continuous Integration and Continuous Deployment (CI/CD) using Jenkins**. The goal is to **automate the entire deployment process** of a Flask application into a Kubernetes cluster, making it seamless and efficient.

Instead of manually executing scripts to build, deploy, and test the application, Jenkins automates the following tasks:

✅ **Fetching the latest code from GitHub**  
✅ **Building the Docker image of the Flask app**  
✅ **Deploying it into Kubernetes (Minikube cluster)**  
✅ **Ensuring the deployment was successful**

By using Jenkins, we enable **automated deployments** whenever changes are pushed to the GitHub repository, making the development workflow faster and reducing manual intervention.

**Project Structure**

my-k8s-app/

│── app/

│ ├── app.py # Flask application

│ ├── requirements.txt # Python dependencies

│ ├── Dockerfile # Docker build instructions

│── k8s/

│ ├── namespace.yaml # Kubernetes Namespace

│ ├── configmap.yaml # Kubernetes ConfigMap for environment variables

│ ├── deployment.yaml # Kubernetes Deployment

│ ├── service.yaml # Kubernetes Service

│── deploy.sh # Manual deployment script

│── jenkinsfile # CI/CD pipeline for Jenkins

│── troubleshoot.sh # Troubleshooting script

**Key Learnings**

**1️.Setting Up Jenkins for CI/CD**

Jenkins is an **automation tool** used for continuous integration and deployment. Here, it is configured to:

* **Trigger builds when new code is pushed to GitHub.**
* **Execute predefined steps to build and deploy the application.**
* **Ensure the deployment is running successfully before marking the build as complete.**

**Steps to Set Up Jenkins**

1. **Install Jenkins** (if not installed already):
2. sudo apt update && sudo apt install -y openjdk-11-jdk
3. wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
4. sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
5. sudo apt update
6. sudo apt install -y jenkins
7. sudo systemctl start jenkins
8. sudo systemctl enable jenkins
9. **Access Jenkins UI** at http://localhost:8080 and configure admin credentials.
10. **Install necessary plugins**:
    * Git Plugin (for fetching code from GitHub).
    * Pipeline Plugin (for writing build steps in a declarative format).
    * Docker Plugin (for building images inside Jenkins).
11. **Generate SSH keys** to connect Jenkins with GitHub:
12. ssh-keygen -t rsa -b 4096 -C "your-email@example.com"
13. cat ~/.ssh/id\_rsa.pub # Add this key to GitHub under "Deploy Keys"
14. **Create a new pipeline project in Jenkins** and link it to the GitHub repository.

**2️.Writing the Jenkinsfile (Pipeline as Code)**

Jenkins uses a **Jenkinsfile** to define the steps it should execute when a build is triggered. The pipeline follows these stages:

pipeline {

agent any

environment {

REPO\_URL = "git@github.com:lazor/kubernetes\_with\_jenkins.git"

WORKDIR = "${WORKSPACE}/my-k8s-app"

VENV\_PATH = "${WORKSPACE}/venv"

DOCKER\_IMAGE = "my-k8s-app:latest"

K8S\_NAMESPACE = "demo-namespace"

}

stages {

stage('Clone Repository') {

steps {

cleanWs()

sh '''

echo "Cloning repository..."

if [ -d "$WORKDIR" ]; then

rm -rf "$WORKDIR"

fi

git clone $REPO\_URL "$WORKDIR"

echo "Repository cloned successfully!"

ls -la "$WORKDIR"

'''

}

}

stage('Setup Python Environment') {

steps {

sh '''

python3 -m venv "$VENV\_PATH"

. "$VENV\_PATH/bin/activate"

pip install --upgrade pip build pytest

'''

}

}

stage('Build Docker Image') {

steps {

script {

echo 'Building Docker Image...'

sh 'cd my-k8s-app/app && docker build -t my-k8s-app:latest .'

}

}

}

stage('Deploy to Kubernetes') {

steps {

sh '''

kubectl apply -f "$WORKDIR/k8s/namespace.yaml"

kubectl apply -f "$WORKDIR/k8s/configmap.yaml"

kubectl apply -f "$WORKDIR/k8s/deployment.yaml"

kubectl apply -f "$WORKDIR/k8s/service.yaml"

kubectl -n $K8S\_NAMESPACE rollout status deployment/flask-app

'''

}

}

}

post {

success {

echo '✅ Deployment successful!'

}

failure {

echo '❌ Pipeline failed. Check logs.'

}

}

}

**Pipeline Breakdown:**

✔ **Stage 1: Clone Repository** → Pulls the latest code from GitHub into Jenkins workspace.  
✔ **Stage 2: Setup Python Environment** → Creates a virtual environment and installs dependencies.  
✔ **Stage 3: Build Docker Image** → Uses docker build to create a container image of the Flask application.  
✔ **Stage 4: Deploy to Kubernetes** → Applies Kubernetes manifests and verifies deployment status.  
✔ **Post Steps:** If the deployment succeeds, it logs **"✅ Deployment successful!"**; otherwise, logs **"❌ Pipeline failed. Check logs."**

**3️.Troubleshooting and Fixes in Jenkins**

**🔴 Issue: Jenkins Failed to Clone GitHub Repository**  
**Fix:** Ensure SSH keys are added to GitHub and correct Git credentials are used.

ssh -T git@github.com

If access is denied, update SSH keys in GitHub settings.

**🔴 Issue: Minikube not Starting in Jenkins**  
**Fix:** Run Minikube with the correct driver:

minikube start --driver=docker

**🔴 Issue: Namespace Not Found Error**  
**Fix:** Ensure correct namespace is used in deployment.yaml and Jenkinsfile.

**🔴 Issue: kubectl rollout status Fails**  
**Fix:** Increase readiness probe delays in deployment.yaml to prevent race conditions.

**4️.Automating Kubernetes Deployment with deploy.sh**

For debugging or running the deployment manually, we wrote a script:

#!/bin/bash

set -e

echo "🚀 Starting Kubernetes Deployment..."

# Start Minikube

if ! minikube status | grep -q "host: Running"; then

echo "🔄 Starting Minikube..."

minikube start

else

echo "✅ Minikube is already running."

fi

# Use Minikube’s Docker environment

eval $(minikube docker-env)

# Build Docker Image

echo "🐳 Building Docker Image..."

cd app

docker build -t my-k8s-app:latest .

cd ..

# Deploy Kubernetes Resources

echo "🛠 Applying Kubernetes resources..."

kubectl apply -f k8s/

kubectl -n demo-namespace rollout status deployment/flask-app

# Get Service URL

echo "🔗 Access application at:"

minikube service flask-service -n demo-namespace --url

**Outcomes**

Now, whenever new code is pushed to **GitHub**, Jenkins automatically:

1. **Pulls the latest code**
2. **Builds the Docker image**
3. **Deploys the application into Kubernetes**
4. **Ensures that the deployment is successful**

This **fully automated CI/CD pipeline** reduces manual work and speeds up software delivery.